Parveen Khurana

124 Followers About Following

Visualizing CNNs

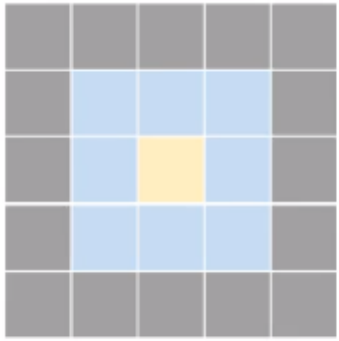
P Parveen Khurana Feb 20, 2020 · 12 min read

This article covers the content discussed in Visualizing CNN's module of the Deep Learning course and all the images are taken from the same module.

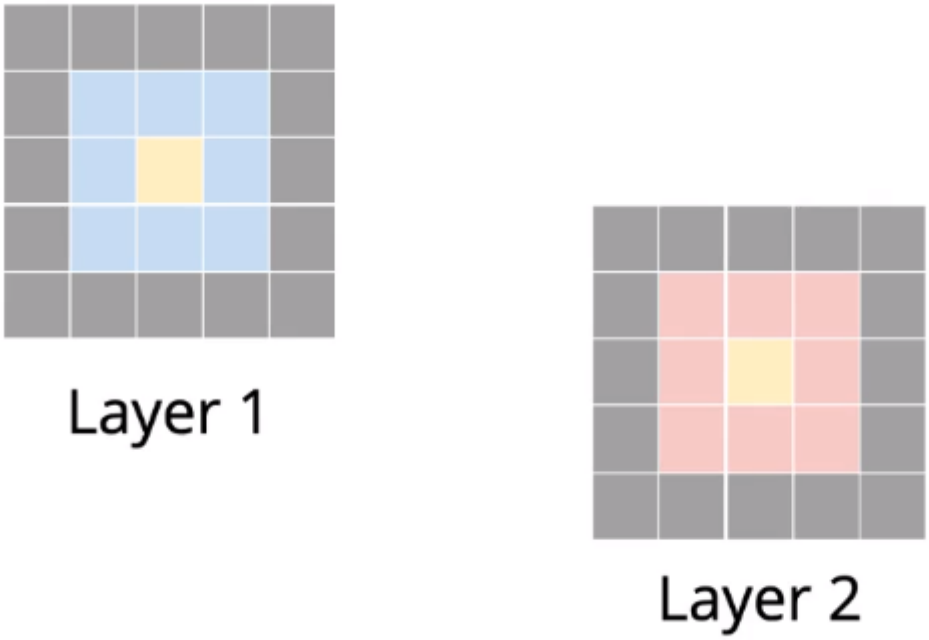
Till now, we have seen several of the standard CNNs. In this article, we try to answer the following question: what kind of images causes certain neurons to fire, what does a filter learn, how good are the hidden representations

The receptive field of a neuron:

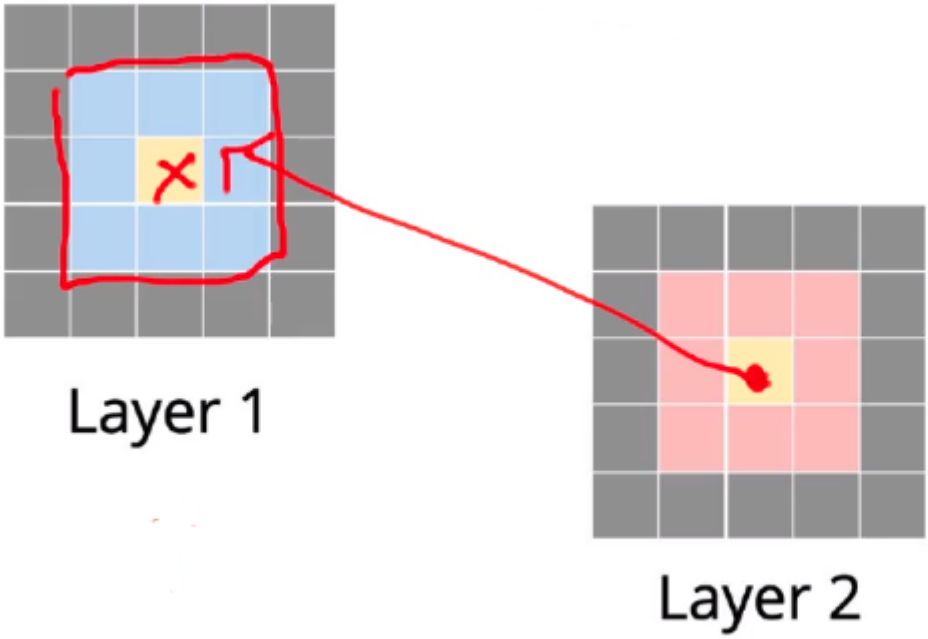
Let's consider a scenario where we have a 3 layered CNN and we use a kernel of size '**3**'. We start at layer 1, our pixel of interest is yellow in the below image and the blue pixels in addition to the yellow pixel represents the position of the kernel.



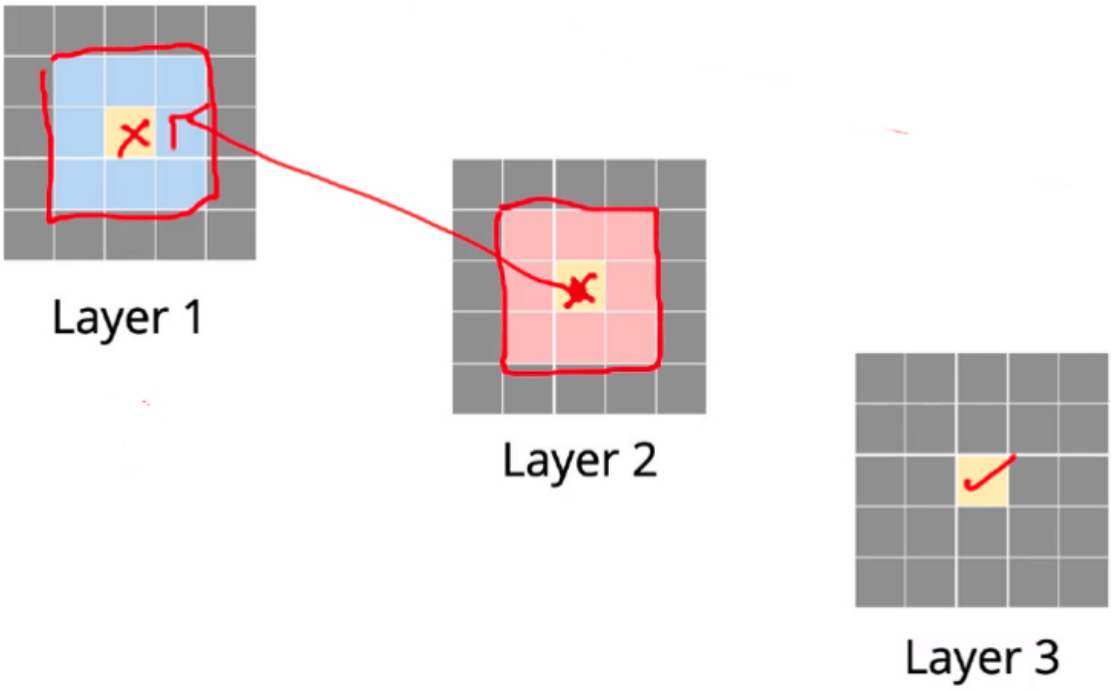
Layer 1



Now, the yellow pixel in layer 2 is computed from a weighted average of '**3 X 3**' neighborhood of the yellow pixel in Layer 1 as depicted in the below image.

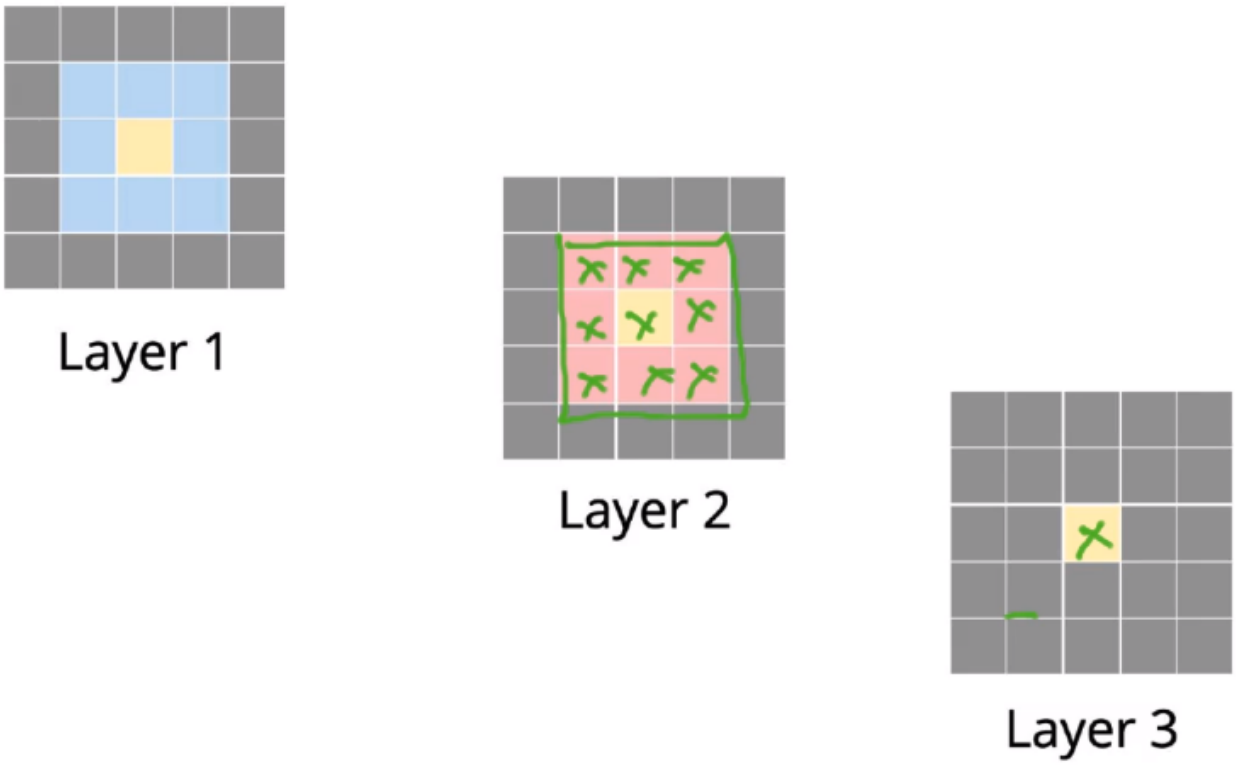


Now, we do the same thing again, we take the '**3 X 3**' neighborhood area around the yellow pixel in Layer 2, take the weighted sum of this '3 X 3' neighborhood and we get

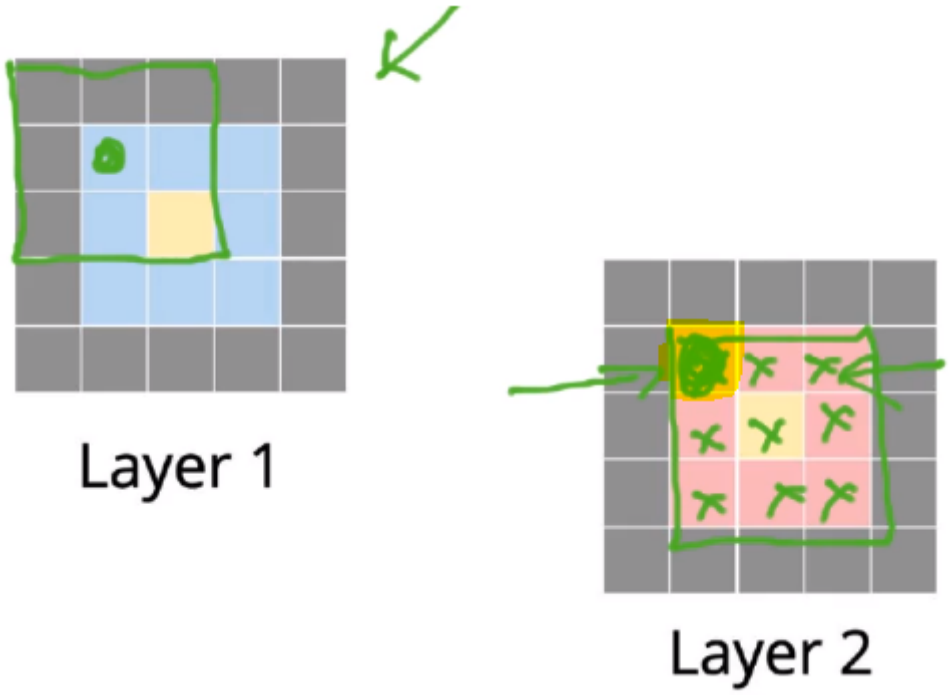


Now the question is what is the **receptive field(how many pixels of the original image are used to compute/influence this pixel) of this yellow pixel in layer 3 as in the original input image(at Layer 1)**.

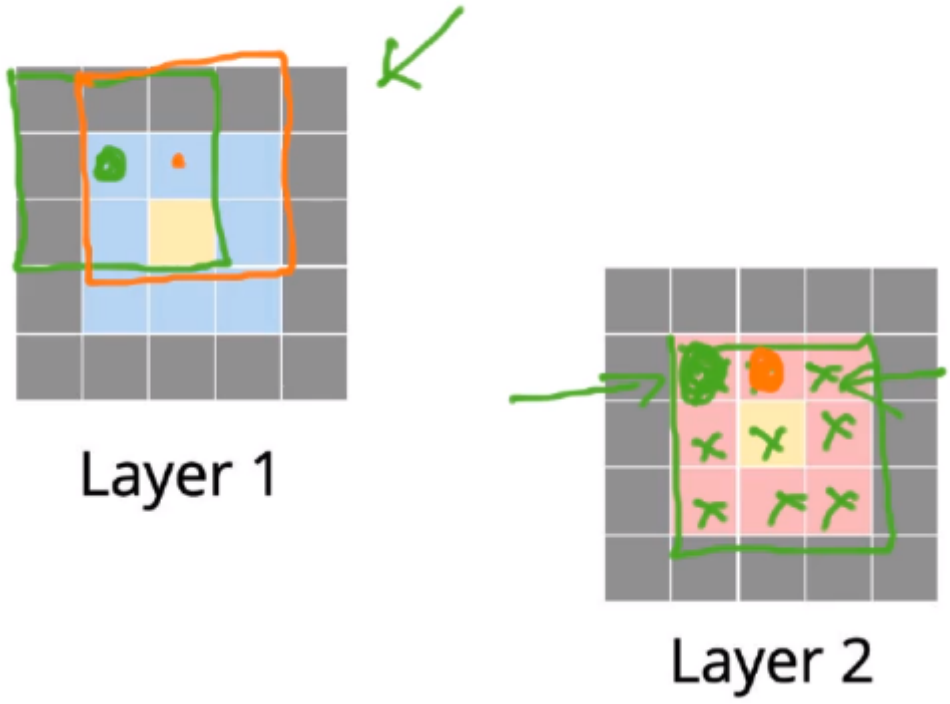
It's clear that this neuron(green marked pixel in layer 3) depends on the '**3 X 3**' neighborhood(or 9 pixels) in the previous layer(Layer 2).

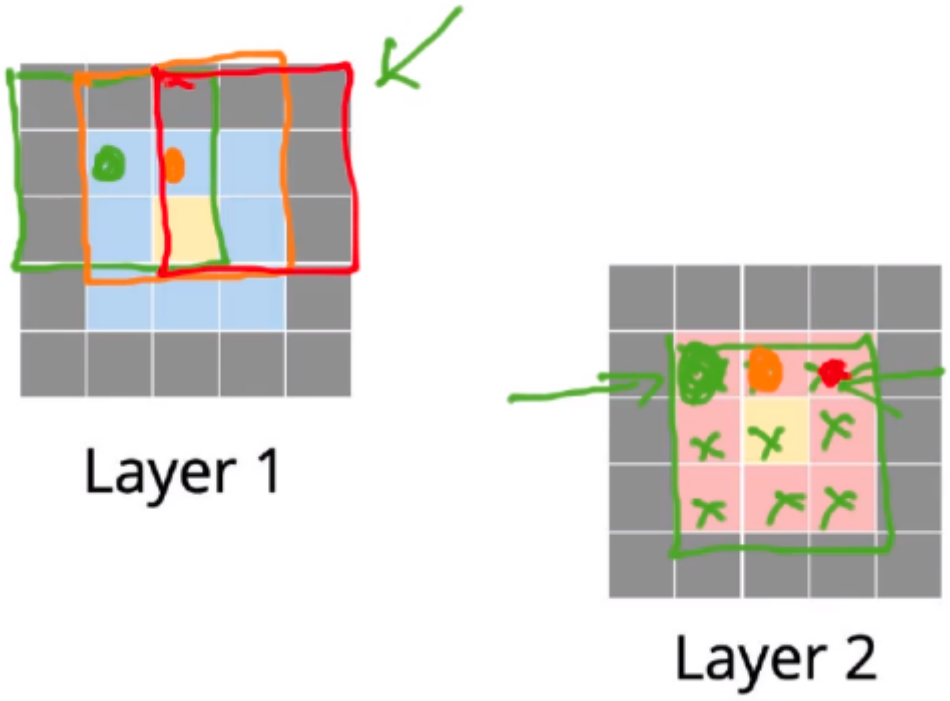


The yellow highlighted pixel in layer 2 in the below image is obtained by placing the '**3 X 3**' kernel over the green bounded region in layer 1 as depicted in the below image:

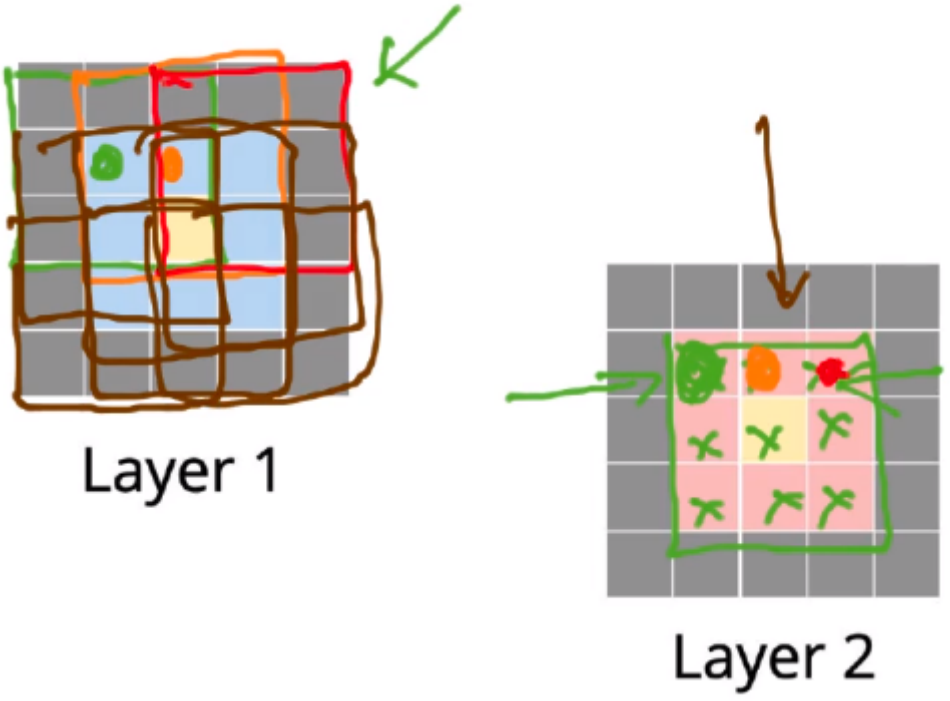


The second pixel(of the '**3 X 3**' neighborhood in layer 2 which influences the central pixel in the layer 3) which is in orange in the below image is obtained by placing the '**3 X 3**' kernel over the orange bounded box in the layer 1 as depicted in the below image:





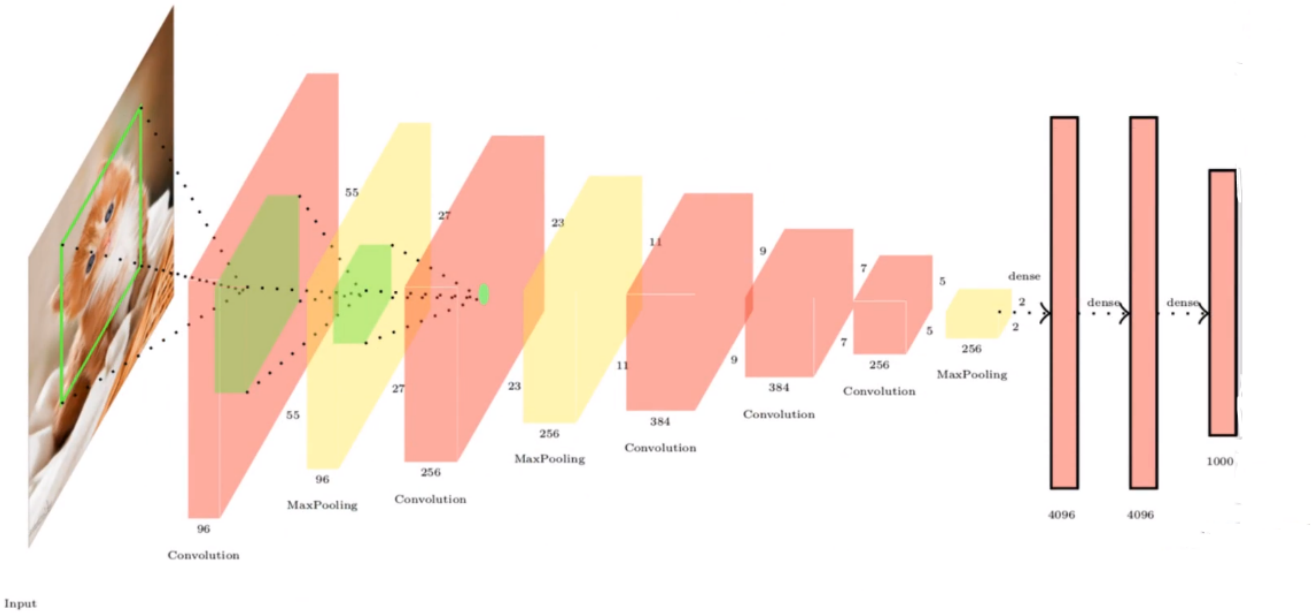
If we continue it like this, we realize that all the 9 pixels of interest in the layer 2 which influences the central pixel in layer 3 are obtained from the below-marked bounding boxes in the layer 1 as depicted in the below image:



layer(Layer 1 in the above case).

As we keep going deeper in the network, the region of influence for the pixels in the deeper and deeper layers, in the original image is going to be larger and larger.

The key thing to note is that **for any pixel in any layer of the convolutional neural network, we could find out what is the region of influence in the original image.**

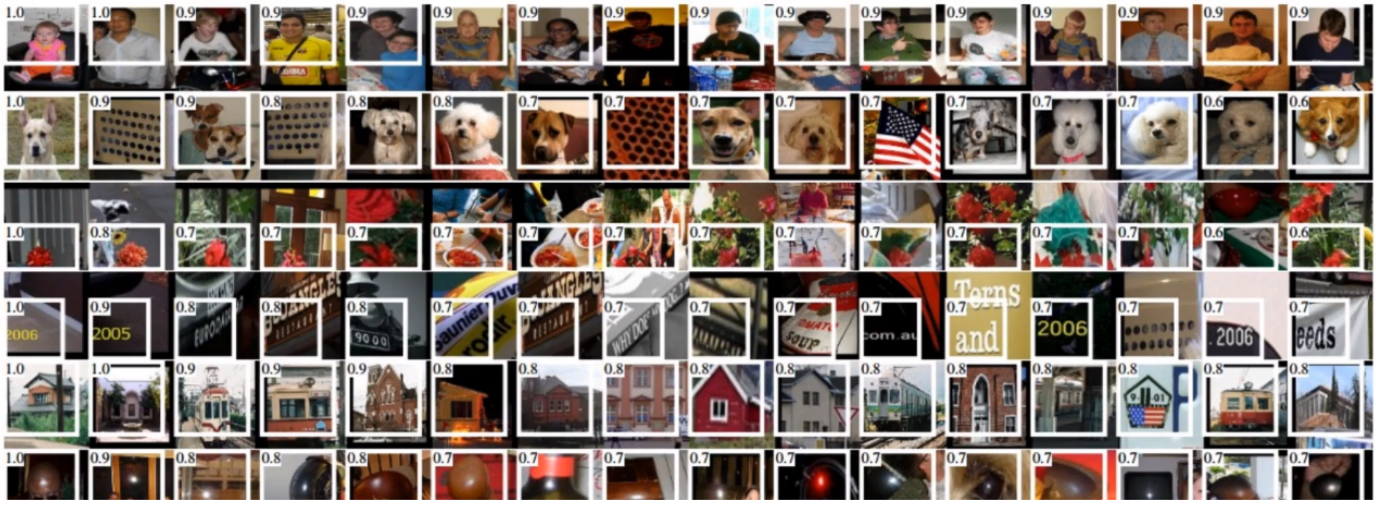


Identifying images which cause certain neurons to fire

We are interested in knowing if different neurons are firing(firing means output is close to 1 in case of sigmoid and tanh and we say a neuron fires when using ReLU activation if it has some large value or greater than some threshold value) for different kinds of inputs/images. It might be the case that some of the neurons in an output volume fire for dog-like input whereas some of the other neurons in the same volume fires for say a human face.

We want to know what neurons are firing for different kinds of input(s). Someone did this experiment, they took a particular layer in a neural network say 3rd layer and then selected 6 neurons in that layer, traced back the receptive field of those neurons in the original image which influences these neurons, they took 1000 images from the dataset and passed it through the network and then for each of the images, they noted the status of all of the 6 neurons whether they fired or not. Then for each of the neurons, say neuron 1, they collected all the sets of the images for which it fired,

the set of all the images which caused neuron 1 to fire and the highlighted region(white bounding box) is the region of the influence in the input image.

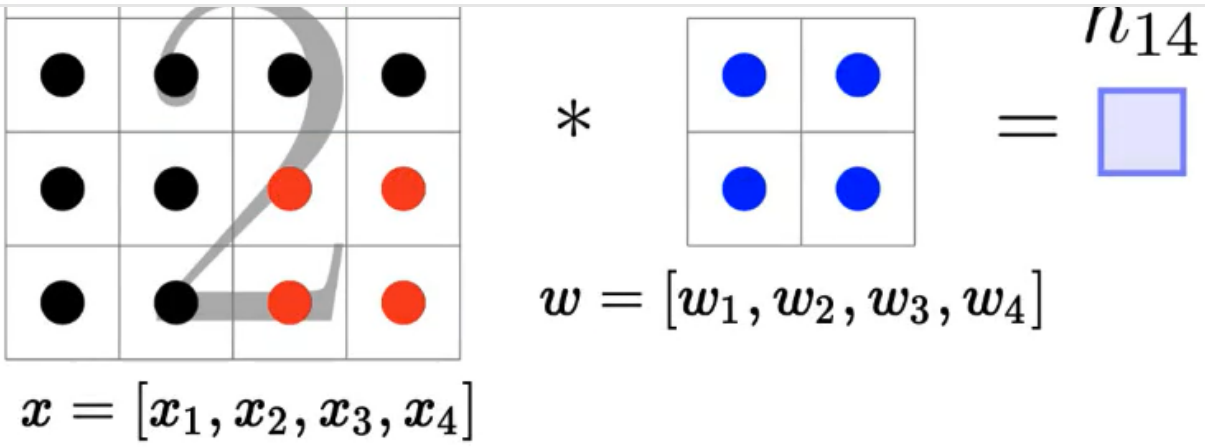


From this, we can say that the neuron 1 is firing whenever there is a human's face in the image. Similarly, say row 2 in the above image represents the set of all the images which caused neuron 2 to fire and as is clear from the above image, neuron 2 gets fired whenever there is a dog's face in the input image. 3rd neuron is firing for flowers mostly red flowers, next neuron is firing for numbers in the image and the 5th neuron is firing for houses in the image and the 6th neuron is firing for shining surfaces in the image.

This kind of analysis helps us to understand whether neurons in different layers are actually learning some meaningful patterns(meaningful pattern means whenever this pattern is given, say a particular neuron would fire and would feed the information to the next layer). So, for all the neurons we want to have a discriminatory power, we don't want all neurons to fire for people's faces, for dog's faces if that is the case that means the neurons are not learning anything meaningful.

Visualizing filters:

The next question that we have in the series is "**what does a filter capture?**". Let's take this question with a situation where we have a '**4 X 4**' input and we are applying a '**2 X 2**' filter on top of that, this filter will slid through the entire image and we are looking at one of the positions where the filter would be eventually(red pixels in the below input image)



Let's say we call these 4 dots/pixels as [**x1, x2, x3, x4**] and similarly, we can think of the weights as [**w1, w2, w3, w4**]. And when we convolve the filter with this input here, we get an output which we call as **h14** in this case. Let's exactly see what **h14** is:

$$\begin{aligned} h_{14} &= w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 \\ &= \mathbf{w}^T \mathbf{x} \\ &\propto cos\theta \end{aligned}$$

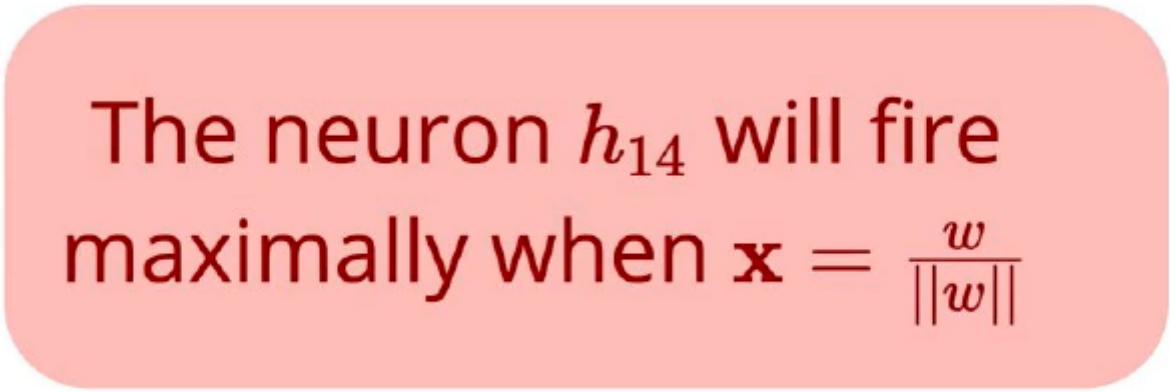
h14 is just the dot product of the two vectors(weight vector and the input pixels, if we treat that as a vector). So, where ever we place our filter over the '**2 X 2**' region of the input, we can say that the output is computed using the dot product of the input vector(pixels => '**2 X 2**' region) and the weight vector.

The dot product is actually proportional to the cosine of the angle between the two vectors, now cosine of the angle between two vectors is given as:

If we place at the filter at certain regions(say certain inputs), let's see for what kind of inputs, will the filter give us the high value or to rephrase, let's see when **h14** will be high:

h14 will be high when the cosine of the angle between two vectors which helps to compute **h14** is high, cosine value would lie between **-1** and **1**, maximum possible value of cosine of an angle would be **1** and that would be possible if the angle between two vectors(inputs '**x**' and weight '**w**' in this case) is actually 0 i.e '**w**' and '**x**' both are in the same direction.

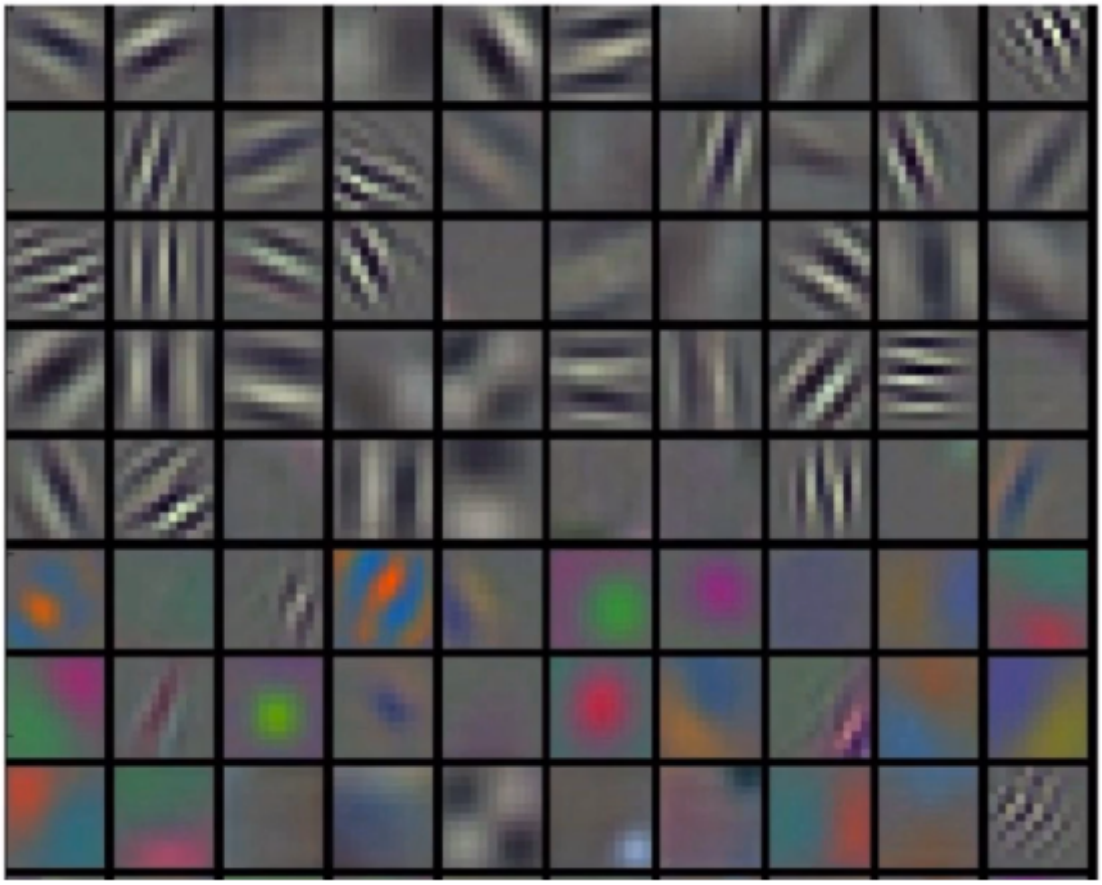
So, **when both the input and the weight vector are in the same direction, the neuron is going to fire maximally**. Another way of this would be that the neuron would fire when the input vector(**x**) is a unit vector in the direction of the weight vector(**w**) or multiple of the unit vector in the direction of '**w**'



We can now start thinking of this filter as an image, we can think of this filter as some patch in the input, whenever the same patch appears in the input, the neuron is going to fire.

We have a filter '**w**', we are going to slide it over the image, whenever we encounter a region in the image which is exactly looking like(pixels values are like some multiple of the '**w**' values) '**w**', that is when the output is going to be maximum. In other words, we can say that the filter is trying to detect regions that are exactly like it in the image. For all other regions in the input which are not the same as the filter, the output is going to be low. So, **the kernel is filtering out for all the regions which are not like it and**

Now if we want to know what patterns the filter is learning, we can just plot out the filter as it is, for example, AlexNet has **96 '11 X 11'** filters in the first layer, each of the squares in the below image represents those '**11 X 11**' filter(total of 80 are shown in the below image out of 96 filters and each of the squares in the below image represents a single '11 X 11' filter)



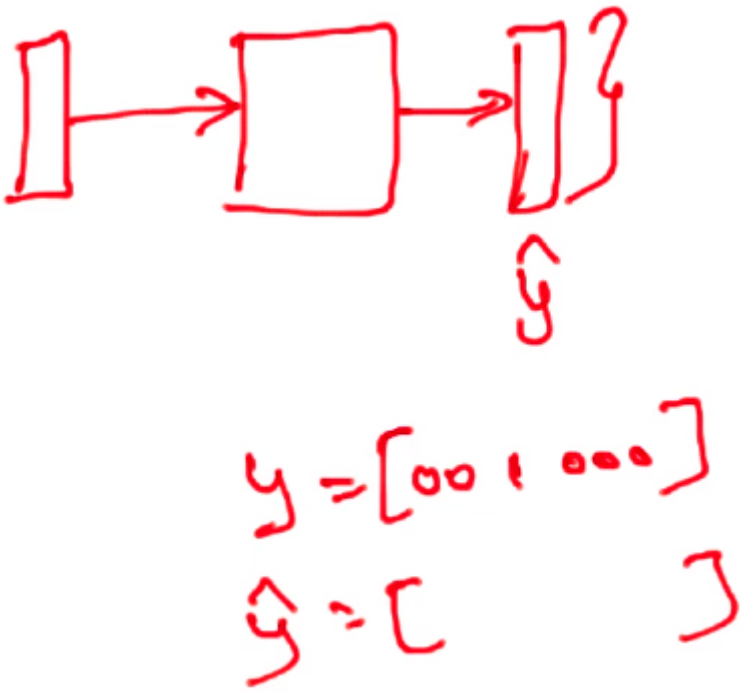
As is clear from the above image, these filters are learning some patterns for example, the very first filter is learning diagonal lines that are going from top left to bottom right, the second filter(1st row, 2nd column) is learning reverse kind of diagonal line compared to the first filter and so on. What this means is that whenever we slide the filter(say the very first one) over the input image, so whenever we have a patch in the image which has a pattern like the diagonal line as in the below image, then the output that we would get is going to be very high or in other words, this filter would produce high outputs for patches in the image which look exactly like it.

filter, we get some idea of the patterns that the filters are learning. We want all the filters to learn different types of patterns, if all of them are learning the same pattern say detecting a vertical line then something is wrong as all filters are redundant in this case, if all the filters are learning different patterns, then they will fire for different patches in the input image and that's exactly what we want.

Occlusion Experiments:

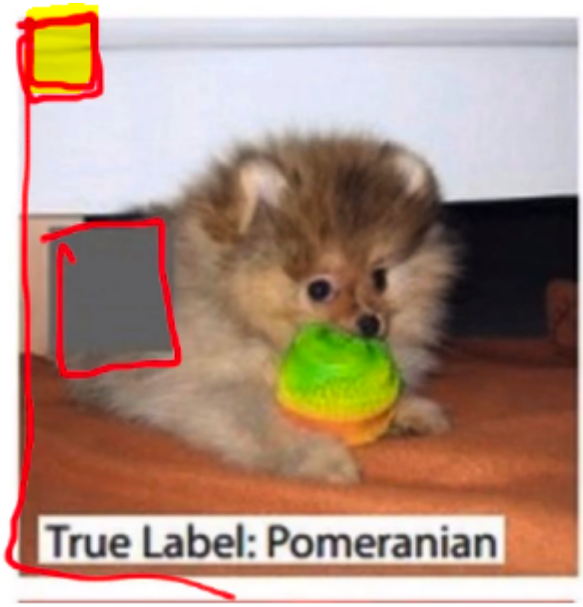
This experiment is to determine which patch in the image contributes maximally to the output. So, let's understand what happens in this:

We have an image as the input, we do a series of convolutions on that and then at the end we have the output layer which is a softmax layer which gives us a probability distribution and we also have the true output which we can represent as a probability distribution in form of a one-hot encoded vector



Let's say the input image is the below one and the model predicted it to be a 'Pomeranian' with a probability of say **0.7**(let say we call it as '**y_hat_1**'). Now what we do is that we create the occulsion patches(red box in the below image)

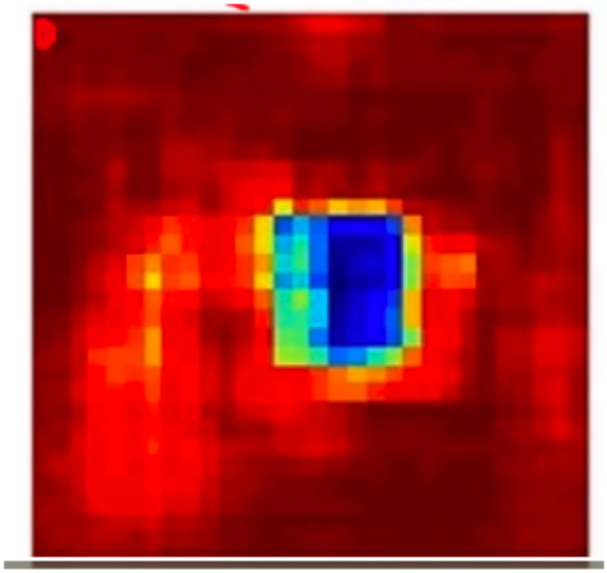
and replace the yellow highlighted portion in the below image with this occlusion patch(i.e with these grey pixels) so that region of the image is no longer there



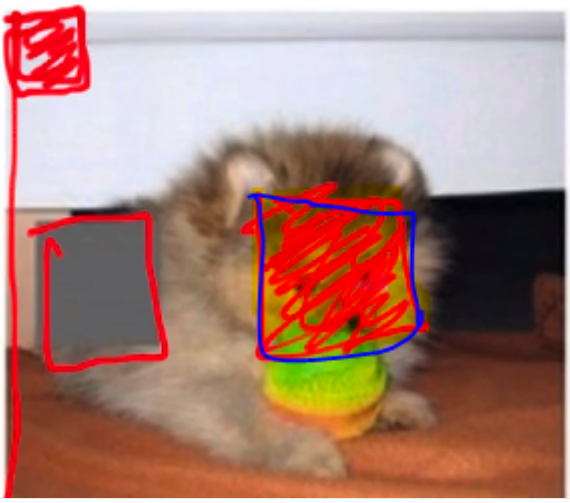
Now the image is a little different compared to the original input, its a dog image(face is clearly visible) just one portion grayed out

We pass this image through the network, compute the predicted distribution, this predicted distribution might change a bit compared to the previously predicted distribution when the entire image was passed through the network as it is and given that we have greyed out the topmost region of the image, the dog is not even present in that portion, so we will expect that the probability would not change much and at max, it would become say **0.69** or would remain **0.7**, we will call it as '**y_hat_2**'(probability distribution after greying out certain portion of the image).

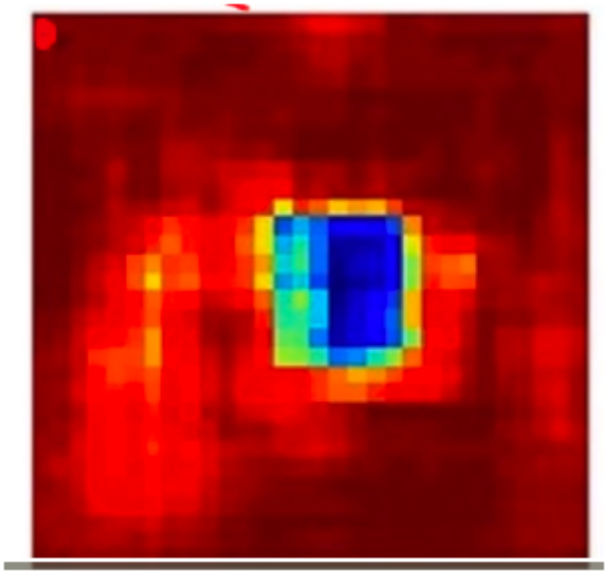
Now we can plot the difference between '**y_hat_1**' and '**y_hat_2**' and we can plot it out as a heatmap(we are plotting the change in the probability/predicted distribution when we grayed out a certain portion of the image).



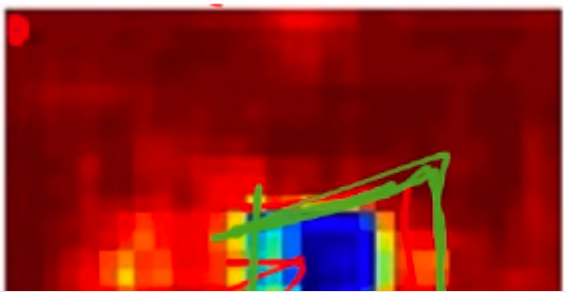
Let's see what would be the case when we apply the patch the dog's face in the image(only the blue bounding box in the below image)



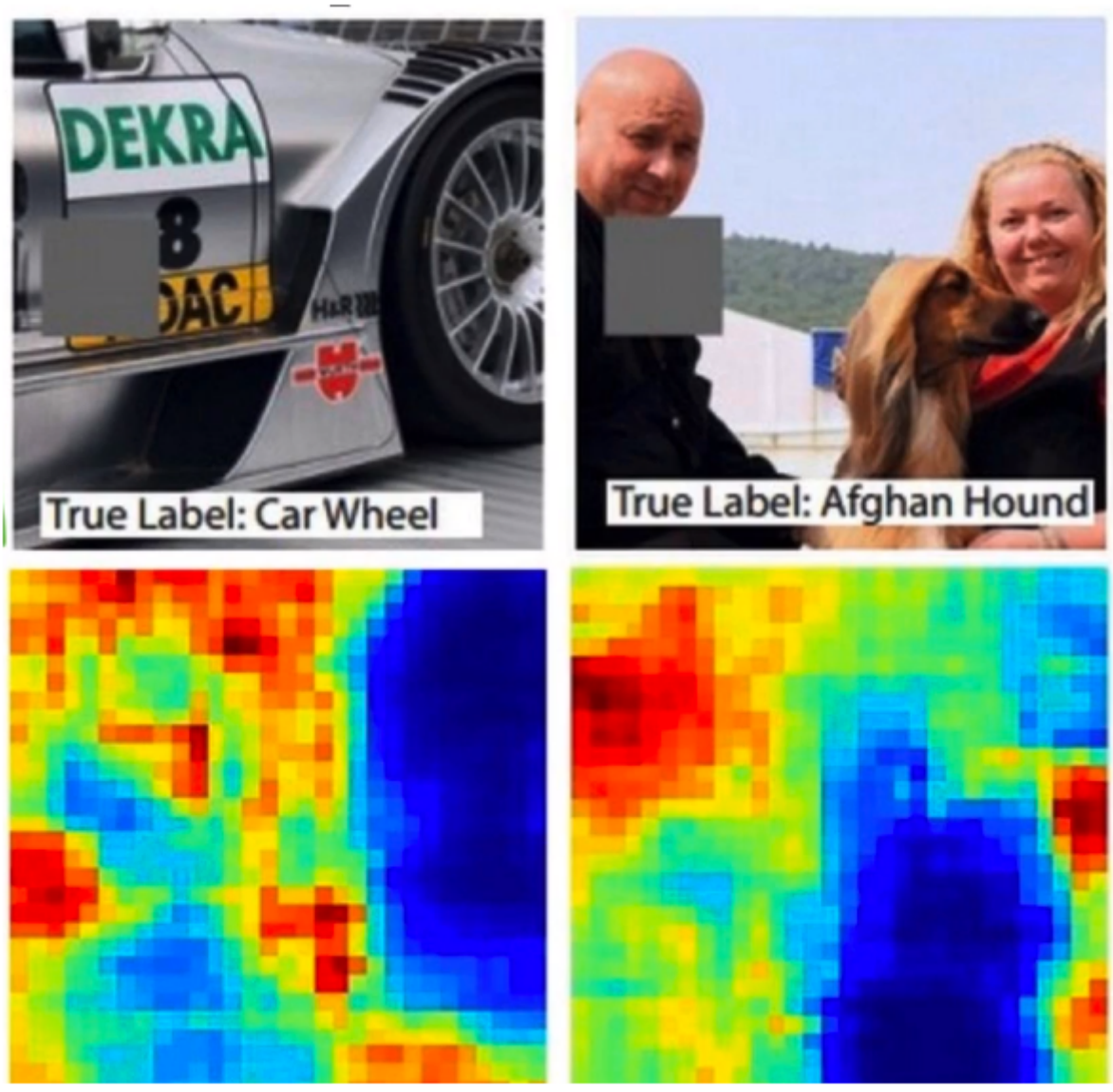
It is very likely that the probability would drop drastically, it would become say **0.2** or somewhere around that because that's the dog's face that we greyed out and it is the most crucial part in the input, so probability would drop drastically and if take the difference between the two predicted distributions(one when the entire image is passed as it is and the other one when we greyed out the dog's face), the difference is going to be high, which is represented by blue shade in the heatmap depicted below



So, **the higher the difference, the more blue shade in the heatmap and the lower the difference, the dark red is the color in the heatmap**. What heatmap tells us is that if we gray out any region in the green bounding box in the below image, then the difference between the two predicted distributions would be high or in other words, the probability of the current class drops drastically and that is intuitive. So, this actually tells that CNN is learning some meaningful pattern, it's learning how to detect the dog based on the face of the dog if we grey that part out the accuracy drops drastically. This tells us that CNN is not learning to predict the dog's class based on the floor in the image, it's actually learning to do that based on the face of the dog, which tells us that CNN has learned something meaningful.



Similarly, we have the below cases:



Deep Learning Convolutional Network Artificial Intelligence Artificial Neural Network

Machine Learning